Performance Research: Current Status and Future Directions

David H. Bailey (LBNL), Bronis de Supinski (LLNL), Jack Dongarra (Univ. of Tenn.), Jeff Hollingsworth (Univ. of Mar.), Paul Hovland (ANL), Shirley Moore (Univ. of Tenn.), Boyana Norris (ANL), Dan Quinlan (LLNL), Daniel Reed (UNC), Allan Snavely (SDSC), Jeffrey Vetter (ORNL), Patrick Worley (ORNL)

22 Feb 2005

1. Introduction

The Performance Evaluation Research Center (PERC) is one of the Integrated Software Infrastructure Centers (ISICs) in DOE's Scientific Discovery through Advanced Computing (SciDAC) program. Its objective is to develop a science of performance analysis on high-end scientific computing system and to engineer practical tools that assist in performance analysis, tuning, and optimization. The original PERC activity had four main focal points: (1) development of benchmarks to serve as targets for performance analysis; (2) development of tools for analysis and optimization; (3) development of models of performance, enabling researchers to predict performance on yet-to-be-built systems; and (4) application of the techniques learned on codes of interest to DOE's Office of Science in general and to the SciDAC program in particular.

PERC recently completed its original three-year dispensation, and is now engaged in an additional two-year activity, known as PERC-2, for which the focus has been modified somewhat. For instance, there is no longer a focus on developing benchmarks—it was felt that we have sufficient benchmark tools for the time being, and instead the resources of the project are focused on updated versions of items (2), (3), and (4) in the preceding paragraph, and on new research in automated performance tuning and fault tolerance. The focus in the tools activity has shifted to new paradigms that scale to much larger numbers of processors—namely 1000 to 10,000 CPUs. The focus in the modeling activity has been to upgrade the tools that already have been demonstrated so that they are easier to use and yield more accurate performance predictions. One additional activity, related to both the tools and applications activities, is the evaluation of existing tools, in the context of using them to analyze specific codes.

In the first section of this paper, we sketch some of the challenges that lie ahead. Technical background for these issues is then presented in subsequent sections, which describe the status of the PERC-2 project and give a few examples of recent research accomplisments. More details on these activities can be obtained on the PERC website http://perc.nersc.gov (see also the list of references at the end of this paper).

2. Looking to the Future

As we look ahead to the future, we see numerous challenges looming in the performance arena. These challenges are principally driven by an expected proliferation of larger, more diverse and more complicated system designs. In this section, we anticipate system architecture changes and how these changes will impact performance modeling and analysis, as well as how we plan to assist application developers cope with these issues.

Most of today's systems consist of either a large number of single-CPU or dual-CPU nodes, connected in a commodity-based interconnection network, or else a cluster of nodes, where each node is a 16- to 64-CPU coherent shared memory RISC multiprocessor. An immediate architectural challenge is presented by the re-emergence of vector-based systems. Further, multicore processors and hyperthreading, in which parallelism is employed within a single processor chip, are rapidly becoming widespread. Other innovations include programmable functional units, programmable network interface controllers, co-processors, field-programmable gate-array systems and other special purpose devices. There may be complicated interactions between architecture and device physics.

One major challenge is that future systems will have many more individual processors. The BlueGene/L system being installed at Lawrence Livermore National Laboratory will soon feature 65,536 dual-core compute nodes; larger systems can be expected in the next few years. Within five years, we almost certainly will see systems with one million individual CPUs, probably organized in a hierarchy of nodes and clusters.

Along with huge numbers of CPUs, we will see designs with adaptive voltage scaling and power consumption. In fact, power consumption is already seen as a critical issue for high-end computing systems. As a result, applications may soon need to optimize not just performance, but also power usage.

As computer systems grow in size, as measured by the number of discrete components, reliability drops, due to the multiplicative effects of failure probabilities. The increasing architectural complexity of these systems, as well as the reliance (at least partially) on commodity-based components, also points to a future where faults will be more frequent. Thus reliability management will need to be integrated into system design, and into performance analysis. In these systems, overall time to solution may be improved by sacrificing raw performance in favor of improved fault tolerance, for example, using real-number codes with some redundancy to implement fault-tolerant matrix operations. In addition to design of the reliability libraries, these systems will require some performance models, in the least, to analyze the trade-offs yielding an optimal program design.

The proliferation and increasing reliance on advances in libraries and system software will also impact performance research. Reliance on precompiled libraries, for instance, defeats performance tools that require complete source code access. On the other hand, perhaps key performance parameters of certain well-known libraries can be stored in a database, so that such analyses can be reused whenever the libraries are used.

In the performance modeling arena, in work to date we have demonstrated a highly accurate (typically within 5% to 10%), semiautomatic modeling capability. One challenge is that even on today's systems, tracing runs generate huge amounts of data and involve a 30X to 100X slowdown in the application being studied. Work to be done includes:

- Retargeting modeling technology to emerging architectures and system designs (this will be a major challenge);

- Improving usability of the modeling tools, so any interested person can use them;
- Automating the process from data collection to model formation;
- Employing "smart" sampling of the running program;
- Modeling the memory hierarchy more carefully, in order to improve accuracy;
- Efficiently capturing the effect of changing application parameters or system scale;
- Studying and evaluating other approaches to modeling.

With the increasing complexity of the architectures and the increasing size of the systems, performance modeling can be a powerful tool in code development. Once a code is written, and identified as performing poorly, it can be very expensive to go back and re-engineer. Performance models could help developers estimate performance while the codes are being developed or modified. This usage of modeling requires that models be easy to generate, easy to update as code evolves, and closely linked to the source code that the developer is generating, but the models need not be highly accurate. Most current modeling methodologies focus on high accuracy predictions for procurement and future technology investigations, not on development. However, performance bounding and modeling assertion tools, along with the source analysis capability in tools such as ROSE, are promising examples of the type of technology needed to use performance modeling in HPC code development. Approximate models linked to the source code will also be useful for pruning the search space in automatic tuning methods.

Most of the currently available performance analysis tools are usable to roughly 1000 processors. However, as emphasized above, future systems will routinely have tens of thousands of processors, and with systems approaching one million individual processors, organized in a hierarchy of nodes and clusters. Thus, it is essential to modify existing tools so that they will be able to handle runs on systems of this scale. This will involve:

- Identifying and removing bottlenecks in existing tools;
- Implementing special versions of tools targeted to very large numbers of nodes;
- Rethinking the existing methodology to gathering and analyzing some types of performance data, (for example, statistical sampling in the SvPablo tool);
- Making use of low-overhead hardware monitoring now available on memory chips, switches and network cards;
- Exploiting advanced hardware and OS features for polling and aggregating monitoring data.

A related challenge is how to handle the huge volumes of performance data that will be generated by running programs on future systems. Some approaches that might work include:

- Dynamic, on-the-fly trace reduction;
- Dynamic, on-the-fly trace analysis;
- Novel large dataset management techniques;
- "Smart" (AI-like) analysis of performance data using multivariate statistical analysis and machine learning techniques.

The last-mentioned technique is particularly intriguing, as it might permit PERC researchers to leverage ongoing research in other areas of large-scale data analysis.

In our discussions with users, and in spite of all the attempts we have made to make our tools as usable as possible, and to educate users in the usage of these tools, it is clear that for most users, the ideal performance tool is one that not only analyzes performance automatically, but also tunes automatically. We believe that the long-term hope for realizing this vision is what we term "generic code optimization." This means semiautomatic tools that generate a search space consisting of numerous options for code tuning, and search for a near optimal solution. We note, for instance, that in a highly parallel environment, the testing of numerous options can be done in parallel, with each processor testing a separate option. Some specific challenges here include:

- Enhancement of automatic code manipulation tools, such as ROSE;
- Automatic run-time algorithmic parameter selection;
- Automatic communication performance analysis;
- Incorporation of performance models into automatic tuning tools;
- Developing new coding techniques and/or programming models that facilitate automatic runtime tuning;
- Heuristics to handle the combinatorial explosion of tuning possibilities.

3. Status Report: Performance Modeling

The objective of our performance modeling activity is to develop technology to produce, with only minor effort, a reasonably accurate performance model, namely a formula or a computer-based tool that can predict the performance of a specific application code on a specific computer system. Some potential applications of performance modeling include: (1) simplifying system procurements at large (or small) computer centers; (2) assisting system designers and operators to better target systems to address certain applications; and (3) helping individual users understand their performance and project performance levels on future systems. Recent work in PERC-2 performance modeling includes:

- Making automated modeling tools more robust, able to characterize large applications running at scale while simultaneously simulating the memory hierarchies of multiple machines in parallel;
- Porting the requisite tracer tools to multiple platforms;
- Improving performance models by using higher-resolution memory models;
- Adding control-flow and data dependency analysis to the tracer tools;
- Exploring a number of new modeling methodologies;
- Using our tools to develop performance models for certain strategic codes;
- Applying our methodology to make several blind performance predictions on mission partner applications, targeting most current system architectures;
- Carrying out error analysis to correct some systematic biases encountered as part of the large-scale blind prediction exercises;
- Adding instrumentation capabilities for communication libraries other than MPI;

- Disseminating the tools and modeling methods to several mission partners, including DoD HPCMO and two DARPA HPCS vendors (Cray and Sun), as well as to the wider HPC community via a series of tutorials.

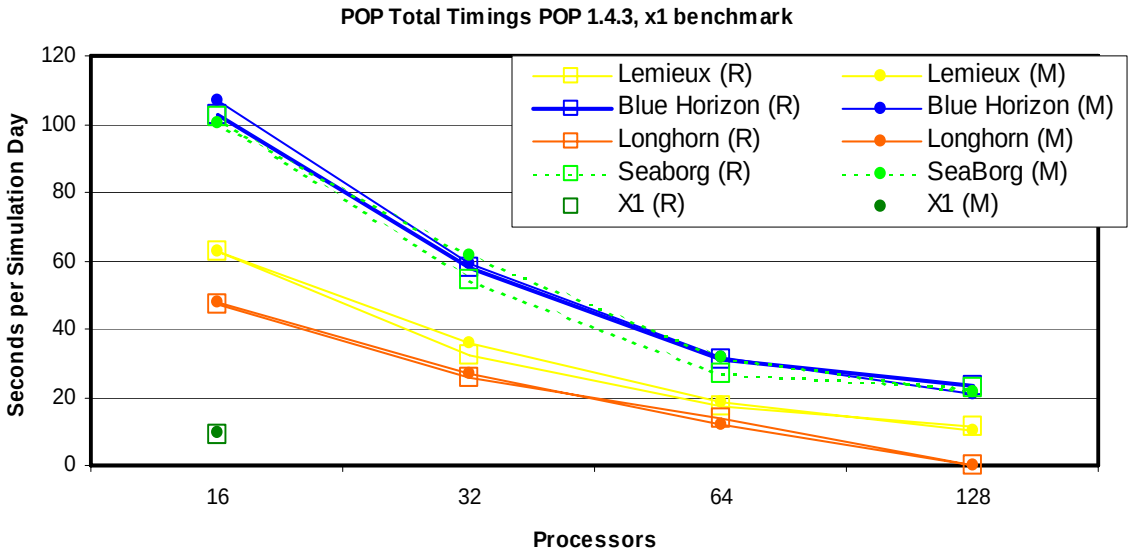
Much of our modeling methodology depends on memory tracing to acquire *application signatures.* Our tools combine these signatures with machine signatures to produce performance predictions for specific codes on specific systems. Because the time dilation of a full memory trace can be very large, PERC-2 researchers have added sampling in time and space to the PERC-2 tracing technology. With these approaches, memory tracing now incurs a slowdown between 30 and 100, an order of magnitude faster than previously achieved, thus rendering the study of long-running applications feasible. Another recent improvement is that the MetaSim Tracer feeds the address stream immediately to cache simulators of several different architectures, thus parallelizing the modeling of the full set of 26 architectures with only minimal additional slowdown.

In other modeling activity, we have developed a set of metrics and performance models for evaluating the effectiveness of various run-time reordering transformations. The model for iteration reordering is a temporal locality hypergraph. The use of this model involved the development of several new iteration reordering heuristics, as well as the reinterpretation of an existing heuristic. This work was presented at the Second ACM SIGPLAN Workshop on Memory System Performance (MSP 2004).

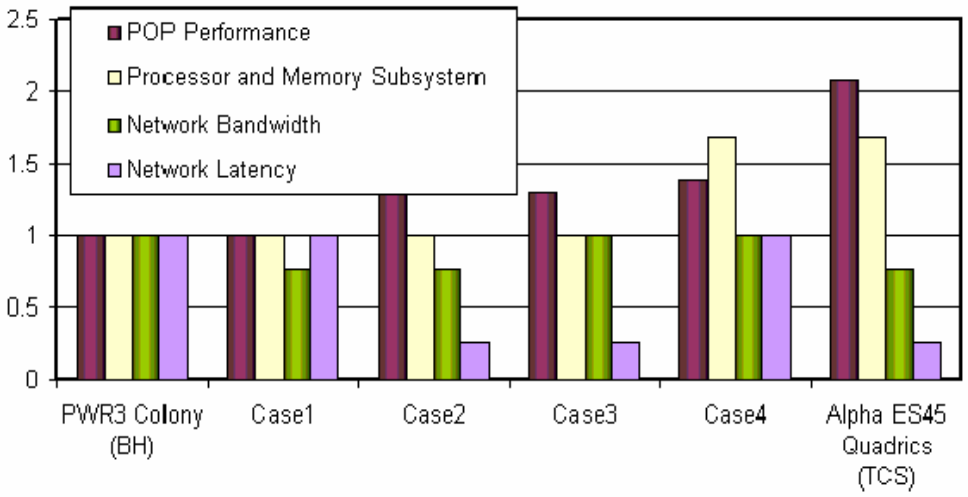
In a separate study, we are studying the use of machine learning techniques to automate application performance prediction across large parameter spaces, such as architectural configurations, parallelism or even application input. Initial results, using cache miss rates as the prediction target, have shown encouraging results. We are currently extending our work to the performance prediction of parallel applications. We have also implemented a number of methods for either interpolating or directly measuring the expected memory performance of real loops that are not all random or all stride one, but some mixture of these as well as other strides. These different options are now built into the PERC-2 Convolver Graphical User Interface (GUI) tool. The investigation of which interpolation works best under what circumstances is still being explored.

We have used our modeling tools to develop a performance model of the SciDAC application POP (Parallel Ocean Program); we are developing one for GYRO (a gyrokinetic fusion simulation program). A sample of POP timings versus performance predictions are shown in the figure below.

We have also applied the framework developed by PERC-2 to the workload of our mission partner, DOD's HPCMO, to make a set of roughly 170 blind performance predictions for TI-05, one of their benchmark programs. These automatically generated performance predictions are *blind* since PERC-2 had access to the machines only via "probes," i.e., the results of low-level benchmarks. These predictions averaged about 20% error. With further analysis we discovered that some "errors" were poorly run real applications, for example, runs in which the system allowed the application to page. Machine reconfiguration should further reduce these errors.



Our performance modeling tools also support "what if" studies on system design, estimating performance changes that would result if changes were made to the system. In the figure below, the baseline case is the Parallel Ocean Program (POP) running on an IBM Power3 system with a Colony interprocessor communication switch. Case 1 indicates how the performance characteristics would change if the switch had the Colony latency but the Quadrics bandwidth; Case 2 indicates Quadrics latency and bandwidth; Case 3 indicates Quadrics latency but Colony bandwidth; Case 4 indicates the Colony latency and bandwidth, but with an Alpha ES640 processor and memory system; Case 5 indicates the Quadrics switch with the Alpha ES640 processors and memory system.



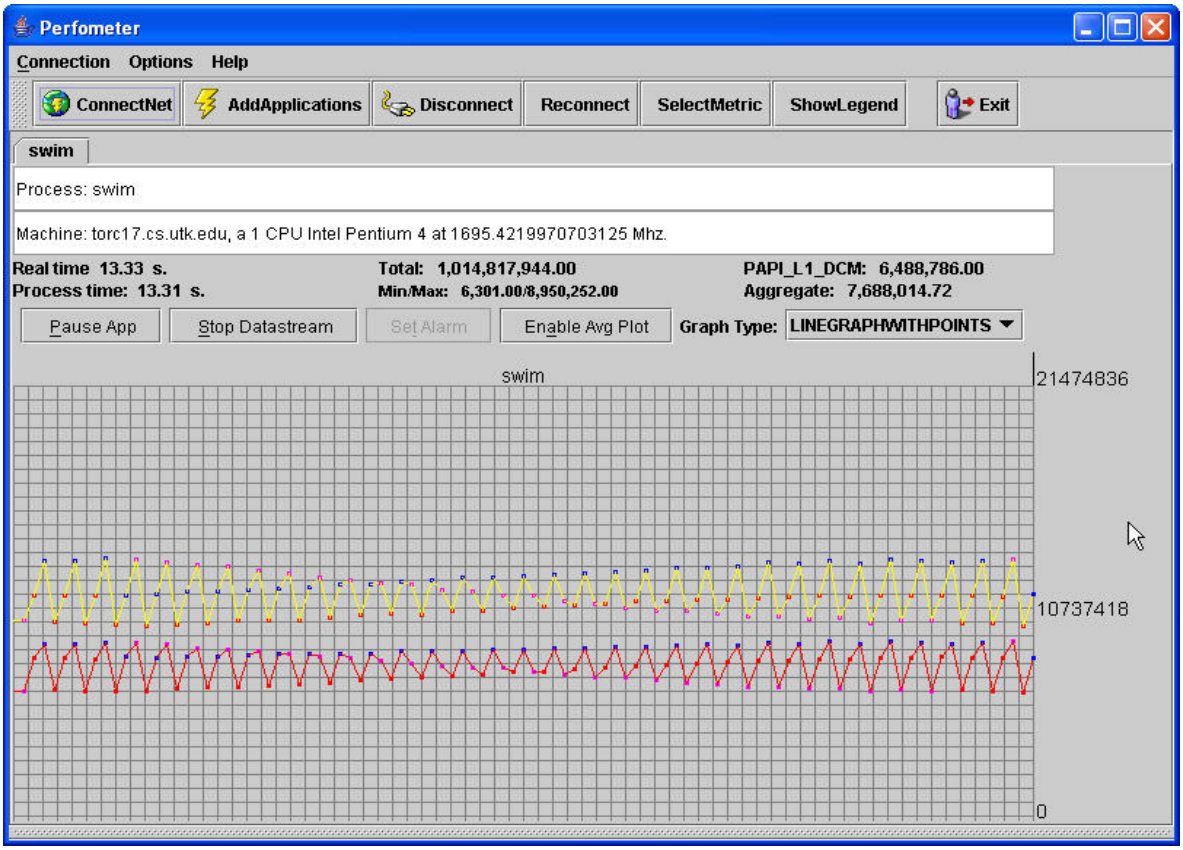
4. Status Report: Performance Tools

The PERC-2 performance tool activity is developing effective and highly usable tools that can analyze performance on high-end scientific systems. Also included here are tools that can make certain required modifications to user codes that will increase runtime performance. Recent PERC-2 work in performance tools includes the following:

- A new release of the PAPI cross-platform hardware performance counter library;
- Enhancements to the KOJAK, SvPablo, and PBT tools;
- Comparative tool assessment in the context of SciDAC applications;
- Addition of several loop optimizations to ROSE compiler infrastructure;
- Addition of new program analysis capabilities and transformation specification mechanisms to ROSE;
- Extensions to the dynamic stream detection tool (dsd) for complex memory reference pattern identification;
- Shared memory application phase detection;
- Automated tuning and search extensions for ATLAS;
- Stratified population sampling for large-scale measurement;
- Fault measurement and power consumption measurement integration;
- Release and enhancements to the mpiP communication profiling tool.

The PAPI library is a hardware performance monitor application programming interface. It provides a standard interface across multiple vendor platforms to the hardware performance counters available on most modern microprocessors. These counters provide a window on the processor by enabling access to relevant information such as operation counts, cache and memory behavior, and branch behavior. PAPI Version 3, a complete rewrite of PAPI, was released in November 2004. New features include much lower overheads on counter start, stop and read operations; full support for named native events; interrupt on overflow and statistical profiling of multiple simultaneous events; memory hierarchy information; shared library mapping; complete thread safety and better thread support. The PAPI library may be used directly by the application programmer. More typically, PAPI is used as the basis for an end-user performance analysis tool, such as the performeter tool illustrated in the figure below, which shows the on-the-fly performance profile (as the code is running) of a shallow water benchmark running on an Intel Pentium 4 system. The yellow curve shows the floating-point operation rate, and the red curve shows the L1 data cache miss rate.

Current work in PAPI focuses on the extension of hardware performance monitoring to off-processor counters, such as those found on memory chips, network switches, and network interface cards. In addition, we are investigating the use of performance counters for estimation of power consumption. While CPU cycles are no longer a precise indicator of power consumption, a combination of several event counts can be used to estimate a processor's power consumption. This estimate can be calibrated with less frequent reading of thermal diodes, the reading of which has much higher latency.



KOJAK 2.0 is a new performance tool suite that collects and analyzes performance data from high performance applications written in Fortran 77/90/95 or C/C++. Performance data are collected automatically using a combination of source code annotations or binary instrumentation and hardware counters. The analysis tools use pattern recognition to automatically convert the raw performance data into performance bottleneck information. For example, a pattern based on the ratio of floating point to memory operations applied to nested loop constructs can automatically detect the occurrence of memory bound loops that might benefit from outer loop unrolling optimizations not always performed by the compiler. Patterns based on communication behaviors can detect inefficiencies such as communication blocking during pipeline refill for wavefront algorithms.

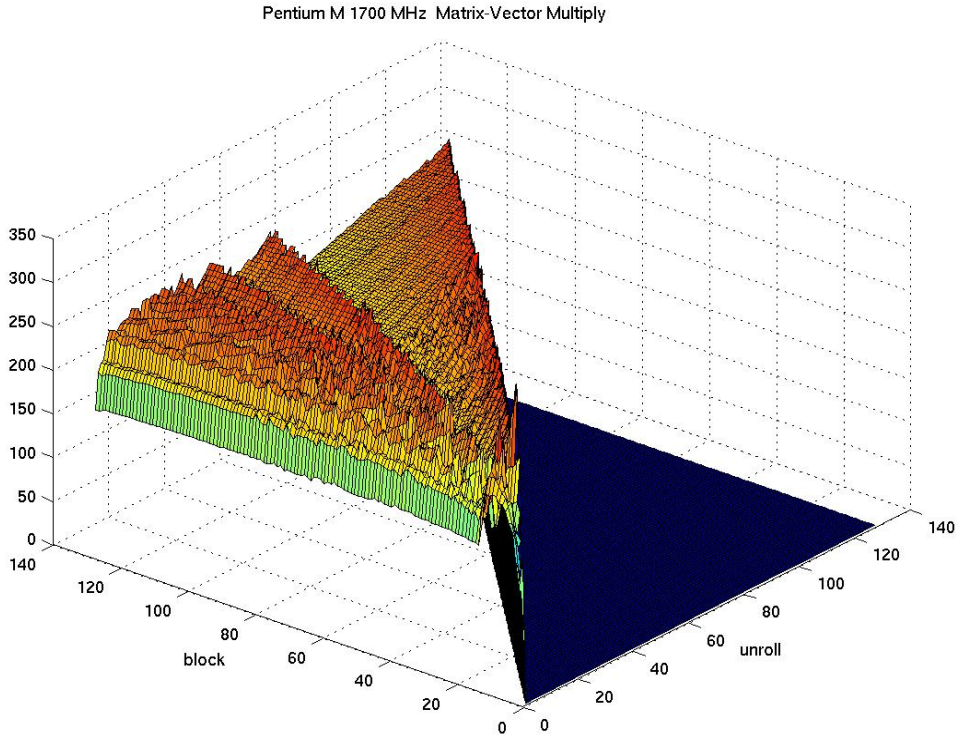
SvPablo is a graphical environment for instrumenting source code, as well as capturing and browsing dynamic performance data at the source level. The PERC-2 activities for SvPablo have centered largely on development of new techniques to enable efficient use of systems containing thousands or tens of thousands of processors. We are evaluating two complementary techniques for large systems: stratified population sampling and power/temperature assessment. We are developing an adaptive performance monitoring system using stratified population sampling techniques, which enables accurate performance measurement of very large systems by sampling the behavior of only a small number of tasks, but with greatly reduced overhead. We are integrating this system into SvPablo, which will present sampling data and error estimates due to sampling.

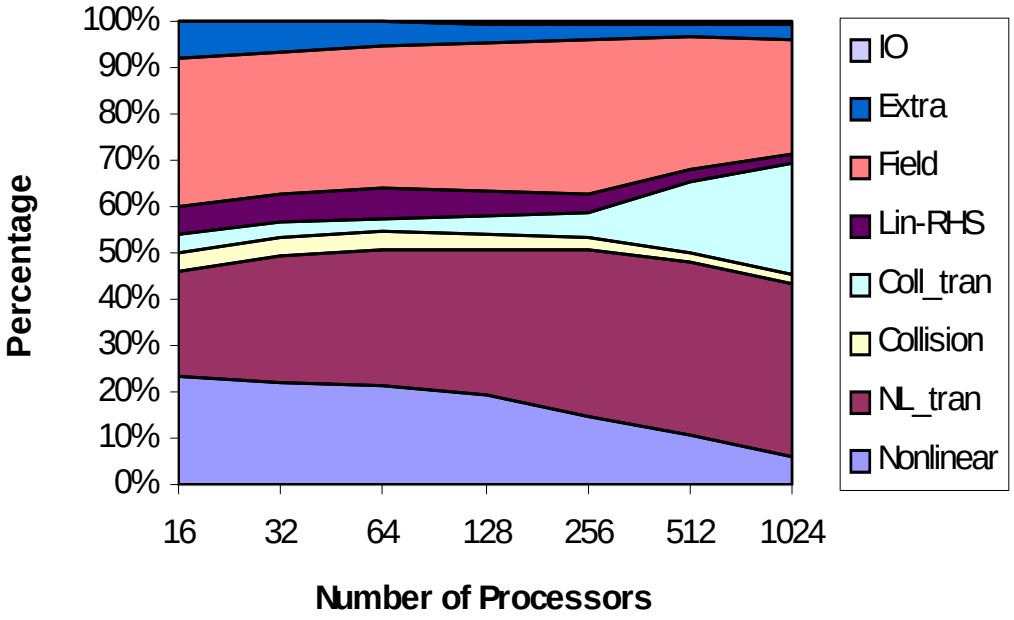
The ROSE compiler project focuses on the optimization of existing scientific applications in C and C++. Using ROSE, source-to-source translators can be easily built to automate customized optimizations more sophisticated or more domain-specific than those provided within vendor compilers. With such translators, high-level abstractions can be used to trigger the generation of low-level platform specific code to provide high performance while still providing the simplicity and productivity of high-level of abstractions for the developer. Using ROSE to automate the generation of efficient low-level code has demonstrated typical improvements of 5-6X for high-level abstractions and up to 15X for some specialized high-level array abstractions. Loop optimizations added to ROSE include support for loop fusion, fission, tiling and unrolling. ROSE now includes whole program analysis support through persistent storage of analysis results in an SQL database. New mechanisms support the specification of transformations and make such transformations more accessible to people without a formal compiler background.

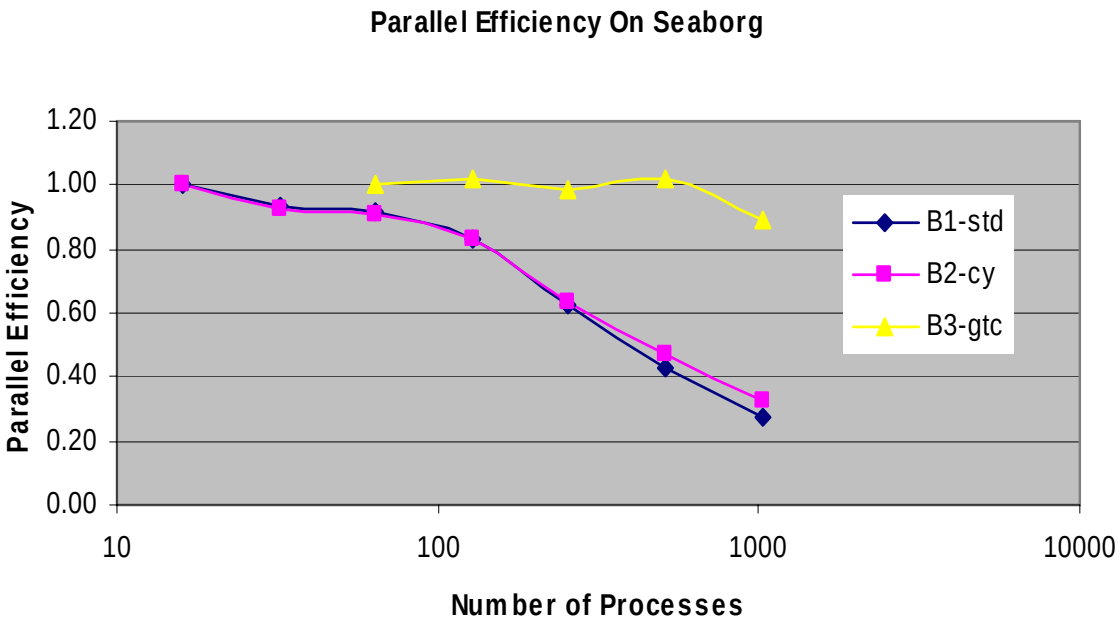
In some potentially far-reaching work, we have pioneered the development of automated performance tuning in the ATLAS system to generate tuned BLAS for specific architectures. This approach generates a search space of algorithms (typically in C or another high-level language), pruned as much as possible by performance models, and then searches the space by compiling and running the algorithms in the search space. Recent work has focused on development of an appropriate search heuristic, based on the Nelder-Mead simplex method, to replace the global search used in ATLAS. A sample of using the simplex method is shown in the figure below. Here the linear algebra routine DGEMV (which performs matrix-vector multiplication) was analyzed to see which of a large number of two-dimensional block options was best. We specify a constraint that the block size should be greater than or equal to the unrolling amount. This allows us to prune a large number of points out of the search space. Then, using the simplex method we can find an "optimal" point in less than 10 minutes (versus 30 hours for the exhaustive search). The figure shows the results of 10,000 runs. 8% of the runs found the true optimum (338 Mflop/s), while on average the value was 87% of the true optimum.

5. Status Report: Tool Evaluation

One of the new activities introduced in PERC-2 is the evaluation of performance analysis tools and methodologies developed both within and outside the project. We perform in-depth performance analyses of SciDAC application codes, applying a number of different tools and methodologies to the same code, then comparing the strengths and weaknesses of the tools and methodologies. Two plots below (one on page 10, the other on page 11) show typical data that we have gathered in this effort. The first shows the parallel efficiency of three variants of the GYRO benchmark (B1-std, B2-cy, and B3-gtc), running on the Seaborg system at LBNL, based on a 16-CPU version as unit efficiency. The second is an analysis of the B1-std benchmark. It shows that the fraction of total runtime spent in "Coll_tran," a key array transposition operation, balloons as the size of the system increases from 256 to 1024 processors. This could be caused by a performance problem in the utility used to implement the data exchange, or simply reflect that the scalability limit is being reached for this benchmark problem. This represents the type of performance question being asked of the performance tools as part of the evaluation.


Pentium M 1700 MHz Matrix-Vector Multiply
350
300
250
200
150
100
50
0
140
120
100
80
block
60
40
20
0
0
20
40
60
unroll
80
100
120
140

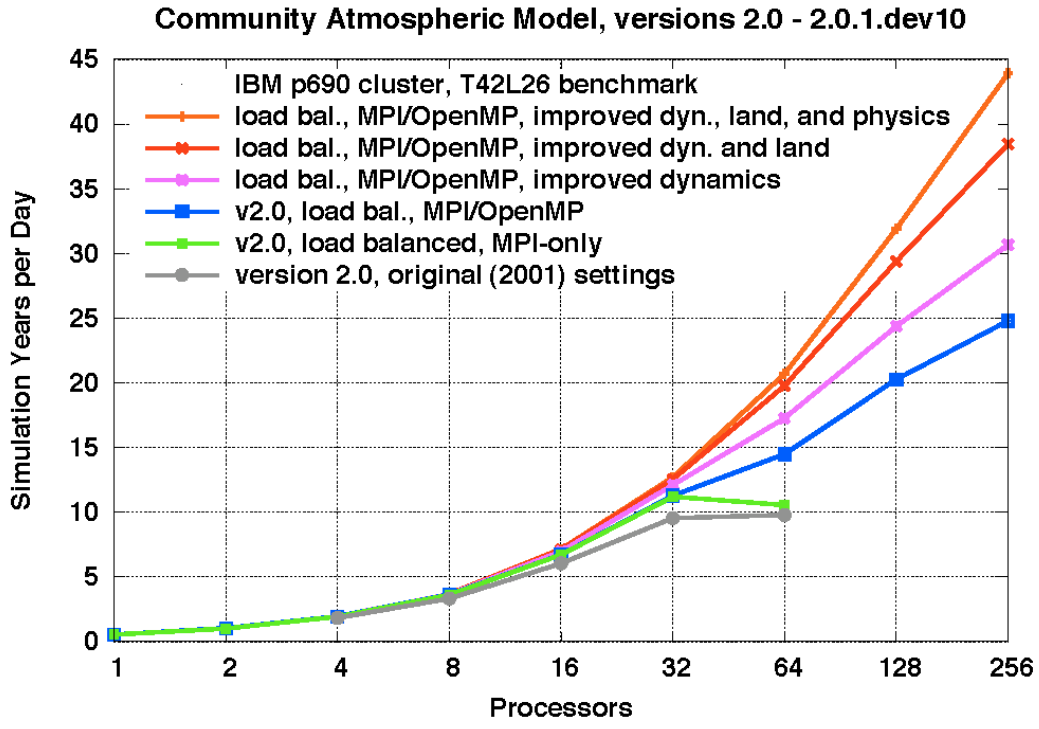

100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
Percentage
16
32
64
128
256
512
1024
Number of Processors
IO
Extra
Field
Lin-RHS
Coll_tran
Collision
NL_tran
Nonlinear

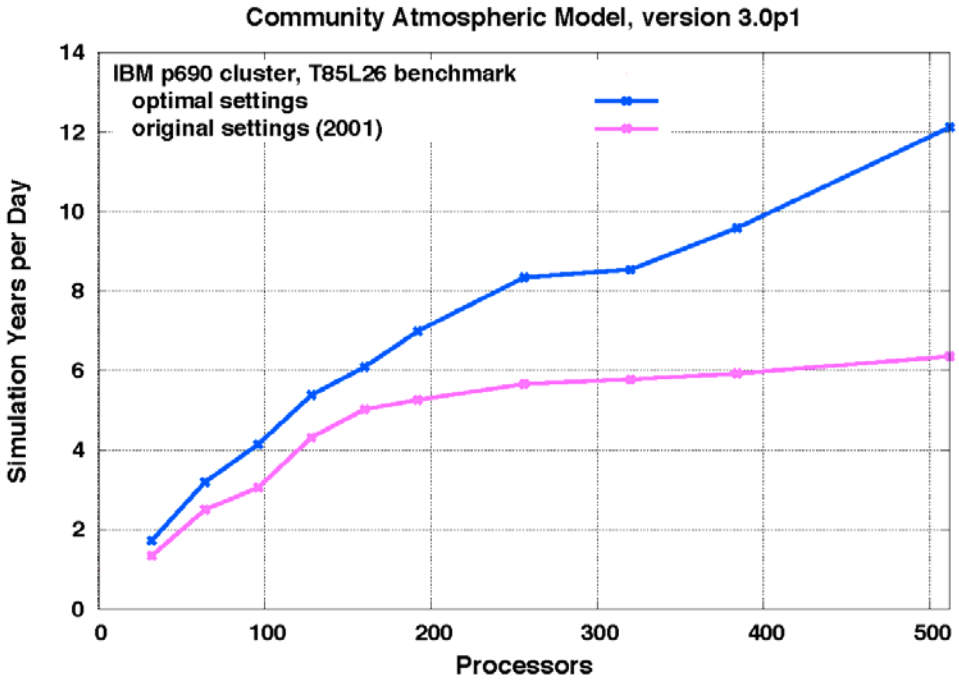


6. Status Report: PERC Collaborations with SciDAC Scientific Projects

PERC continued to work closely with select SciDAC application projects and ISICs. Specifically we have assisted Collaborative Design and Development of the Community Climate System Model for Terascale Computers (CCSM) in determining optimal algorithmic settings and other code optimizations on important Office of Science platforms. We have analyzed and modeled applications of the Plasma Microturbulence Project (PMP), Shedding New Light on Exploding Stars: Terascale Simulation of Neutrino-Driven SuperNovae and Their NucleoSynthesis (TSI), Terascale Optimal PDE Simulations (TOPS), the Terascale Simulation Tools and Technology Center (TSTT) and Enabling Higher Performance for the Lattice Quantum Chromodynamics (QCD).

PERC researchers helped determine optimal algorithmic settings for the Community Atmosphere Model (CAM), a key component of the CCSM, when running the Intergovernmental Panel on Climate Change (IPCC) scenario runs on the IBM p690 cluster at ORNL, thus accelerating the completion of this milestone. Optimization studies were also updated for the standard benchmark problems on the following systems as CAM was ported or as the systems changed: (1) IBM p690 cluster, especially as the SP Switch2 network was replaced by the HPS interconnect and as support for task and memory affinity became available; and (2) Cray X1. Similar studies are ongoing on the SGI Altix, Cray XD1, and the Cray XT3. PERC researchers also completed optimization studies for the Parallel Ocean Program (POP), another CCSM component model, on the Cray X1 and the p690 cluster. The graphs below show results of tuning performed based on recommendations from PERC researchers. The first is based on an earlier version of the CAM code, and the second is based on a recent version.





For PMP, PERC researchers have applied Active Harmony, a software tool supporting distributed execution of computational objects, to GS2, a gyrokinetic turbulence simulator. The result of this effort has been a 2.3 to 3.4X speedup of GS2 for a common configuration used in production runs. PERC researchers also studied the performance of GYRO, an Eulerian gyrokinetic-Maxwell solver. In these studies, scaling behavior and

the impact of task and memory affinity were examined on both the IBM SP3 system at NERSC and the IBM p690 cluster at ORNL. Finally, PERC researchers created a performance model of GYRO and validated it on four platforms at ORNL (Cray X1, SGI Altix, IBM SP3, and IBM p690). This model will provide insight into understanding the performance on new platforms, such as the Cray XD1 and the Cray XT3.

For TSI, PERC researchers ported and optimized the EVH1 hydrodynamics code on the Cray X1, achieving excellent performance for large problems. The EVH1 performance analysis was completed for up to 256 processors on all current target platforms. For the ZEUS-MP CFD code, researchers completed a timing analysis on all target architectures except the SGI Altix for up to 128 processors. ZEUS-MP has also been instrumented with SvPablo on the p690 cluster, in preparation for detailed performance analyses.

PERC researchers analyzed the performance of a TOPS-TSTT mesh smoothing application using the Performance Bounding Tool (PBT), the TAU performance analysis system, and the Performance Application Programming Interface (PAPI). The three dominant phases of the application are gradient computation, Hessian computation, and sparse matrix-vector multiply. After applying runtime reordering transformations, the sparse matrix-vector multiply achieves 90% of the peak performance imposed by the memory bandwidth limit. The gradient and Hessian computations, as optimized by PERC researchers, are not memory bandwidth limited and achieve approximately 25% of machine peak. We are currently investigating the performance limiters of these computations. This application was also modeled using the Metasim Convolver. The convolver accurately predicted the two-fold performance improvement achieved through runtime reordering.

Based on a performance model of QCD's MILC application created by MILC developers, PERC researchers have conducted performance analyses communicate-bound and compute-bound cases on Pentium-3 Linux clusters. We used SvPablo to collect detailed performance data on different aspects of the code, and identified the key fragments that affect code performance. The performance study results have been presented at SciDAC QCD meetings and have helped the QCD community to understand MILC's performance behavior on different execution platforms and configurations.

References

We list here a few sample recent references. Additional reports and papers that we have produced are available at http://perc.nersc.gov.

1. D.H. Ahn and J.S. Vetter, "Scalable Analysis Techniques for Microprocessor Performance Counter Metrics," *Proc. SC 2002*, 2002.
2. David H. Bailey and Allan S. Snavely, "Performance Modeling: Understanding the Present and Predicting the Future," *Proceedings of SIAM PP04*, 2005, to appear. The full paper is available at http://crd.lbl.gov/~dhbailey/dhbpapers/dhb-perf-model.pdf.

3. Laura Carrington, Nicole Wolter, Allan Snavely, and Cynthia Bailey Lee, "Applying an Automated Framework to Produce Accurate Blind Performance Predictions of Full-Scale HPC Applications," *DOD User Group Conference 2004*, Williamsburg, June 2004.
4. Zizhong Chen and Jack Dongarra**. "**Numerically Stable Real-Number Codes Based on Random Matrices**,"** 2004 IEEE Information Theory Workshop, San Antonio, Texas, October 24-29, 2004.
5. I-Hsin Chung, Jeffrey K. Hollingsworth, "Using Information from Prior Runs to Improve Automated Tuning Systems," *Proceedings of SC'04*, Nov. 2004.
6. I-Hsin Chung, Jeffrey K. Hollingsworth, "Automated Cluster-Based Web Service Performance Tuning," *Proceedings of IEEE Conference on High Performance Distributed Computing (HPDC)*, June 2004.
7. Thomas H. Dunigan, Jr., Jeffrey S. Vetter, James B. White III, and Patrick H. Worley, "Performance Evaluation of the Cray X1 Distributed Shared Memory Architecture," IEEE *Micro* (in press).
8. Jeff Hollingsworth, Allan Snavely, Simone Sbaraglia, K Ekanadham, "EMPS: An Environment for Memory Performance Studies," *IPDPS 2005, Next Generation Software Workshop*, to appear.
9. Shirley Moore, Felix Wolf, Jack Dongarra, and Bernd Mohr, "Improving Time to Solution with Automated Performance Analysis", Second Workshop on Productivity and Performance in High-End Computing (P-PHEC), held in conjunction with the 11th International Symposium on High-Performance Computer Architecture (HPCA-2005), San Francisco, February 13, 2005.
10. Daniel Quinlan, Markus Schordan, Qing Yi and Bronis de Supinski, "Semantic-Driven Parallelization of Loops Operating on User-Defined Containers," *16th Annual Workshop on Languages and Compilers for Parallel Computing* (LCPC) College Station, TX, USA, October 2-4, 2003.
11. Michelle Mills Strout and Paul D. Hovland, "Metrics and Models for Reordering Transformations," in *Proceedings of the Second ACM SIGPLAN Workshop on Memory System Performance (MSP)*, pages 23-34, June 8, 2004.
12. Mustafa M. Tikir, Jeffrey K. Hollingsworth, "Using Hardware Counters to Automatically Improve Memory Performance, *Proceedings of SC2004*, Nov. 2004.
13. J.S. Vetter and M.O. McCracken, "Statistical Scalability Analysis of Communication Operations in Distributed Applications," *Proc. ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPOPP*), 2001.
14. J.S. Vetter and P. Worley, "Asserting Performance Expectations," *Proc. SC 2002*, 2002.
15. P. H. Worley, "Federation Performance on the ORNL p690 Cluster," ScicomP 10, Texas Advanced Computing Center, Austin, Texas, August 12, 2004.
16. P.H. Worley, "The Performance Evolution of the Parallel Ocean Program on the Cray X1," Performance and Productivity of Extreme-Scale Parallel Systems, LACSI 2004, Eldorado Hotel, Santa Fe, New Mexico, October 12, 2004.
17. Qing Yi and Dan Quinlan, "Applying Loop Optimizations to Object-Oriented Abstractions Through General Classification of Array Semantics," In the *17th International Workshop on Languages and Compilers for Parallel Computing*, West Lafayette, IN, USA, September 2004.